Name: _____

Student Number: 130772

University of Saskatchewan
Department of Computer Science

CMPT 115-04

Midterm Examination – open Notes/Text
Feb. 15, 2001

Total Marks: 80 Time: 80 Minutes

Answer all of the questions in the spaces provided in this exam paper. If you don't have enough space, write on the back of the page, indicating clearly that your answer is continued there. Be sure to pace yourself according to the marks allotted to each question ... good luck!!

1) True/False Questions (5 Marks)

F A Java class may have only one constructor method.

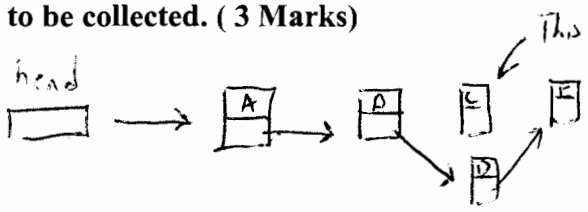
T The subclass cannot access private methods of the parent class

T All java classes descend from a single class.

F T A java class may be defined so that it extends more than one parent class

F T When you call super.clone you are calling the clone method defined in the class Cloneable.

2) **What is garbage collection ? Draw a pointer diagram to illustrate "garbage" to be collected. (3 Marks)**



Garbage collection is picking up objects that are no longer useful.

3) **What is polymorphism and why is it useful ? Give a brief example of polymorphism using a collection class. (3 Marks)**

4) **What is the difference between a Bag and a Sequence ? (2 Mark)**

A Bag has no specific order and a Sequence does. Items in a Bag go anywhere, but in a Sequence they are in a logical order

5) **Write the necessary java code to create a 3 dimensional array that will store 1000 items of the class Object. All 3 dimensions of the array should be equal in size (2 Marks)**

```
Object [][][] holder;
holder = new Object [10][10][10];
```

The following two questions (6 and 7) may or may not have Java code that is incorrect. Incorrect code will result in a compile-time error. State whether the code in each of these questions is correct or incorrect. If the code is incorrect, state what is wrong with the Java code and suggest a correction.

6) (3 Marks)

```
IntArrayBag a, b;

a = new IntArrayBag();

a.add(5);
~~a.add(6);~~ a.add(6);
b = (IntArrayBag) a.clone();
```

This code is incorrect. IntArrayBag b has to call a constructor first before ~~it can be added~~ an int can be added. By changing b to a in the fourth line this code will work.

7) (3 Marks). Assume that we are using the JCL Vector and Stack classes. Recall that Stack inherits from Vector. The method push(Object obj) is a method of Stack. add(Object obj) and remove (Object obj) are methods of the class Vector.

```
Stack s;
Vector v;

s = new Stack();
s.push (new Integer(5));
s.push(new String("cheers"));
s.add (new Integer(4));
s.remove (new Integer(5));

v = s;
```

This code is correct.

8) Assume the following method is invoked. What is the output (6 Marks).

```
int x = 5;
int y = 9;
Integer z, w;

w  = new Integer(5);
z = w;
System.out.println(z==w);
System.out.println(y==x);
y -= x;
System.out.println(y);
System.out.println(y++);
z = new Integer(x);
System.out.println(z==w);
System.out.println(z.equals(new Integer(5)));
```

x= 5
z. 5
y= 4 5

True
False
4
4
False
True

For the following questions use this class declaration and instance variables. Assume that each student is uniquely identifiable by studentNo.

```
class Student implements Cloneable {

private int studentNo;
private string firstName;
private string lastName;
private int classesCompleted;

private int[] grades;
...
}
```

9) Write an **equals(Object obj)** method for class Student **(4 Marks)**

```
public boolean equals (Student s1) {
    if(
      return ( this.studentNo == s1.studentNo )
}
```

Answer

```
public boolean equals (Object obj) {
    if (obj instanceof Student){
      Student otherStudent = (Student)obj;
      return (otherStudent.studentNo ==
              studentNo);
    }
    else
      return false;
```

10) Write a **clone()** method for this class Student**(5 Marks)**

```
public Object clone () {
    Student answer;
    try {
        answer = (Student) super.clone();
    }
    catch (CloneNotSupportedException e) {
        throw new RuntimeException ("This class does not implement Cloneable.");
    }
    answer.grades = (int[] grades.clone());
    return answer;
}
```

11) Write a **toString()** method for the class Student. The toString() method should return a string. When printed the string should produce output similar in content and format to the example below. Assume that the number of classes completed varies from student to student. If no classes are completed do not print an average. Simply state "no classes completed" instead of the last line of output indicated in the example below. **(7 Marks)**

```
Student No: 123456
Name: Smith, John
Grades:
78
77
92
84
76
81
Student Completed 6 classes with an average of 81.3333
```

☆ Should have returned a string instead of ~~having~~ using println() statements.

```
public void toString() {
    System.out.println("Student No: " + this.studentNo);
    System.out.println("Name: " + this.lastName + "," + this.firstName);
    if (this.classesCompleted == 0)
        System.out.println("no classes completed");
    else {
        System.out.println("Grades: ");
        ~~int average~~ int sum = 0;
        for (int i = ~~this.classesCompleted~~ 0; i < this.classesCompleted; i++) {
            ~~System~~ int k = grades[i];
            System.out.println(k);
            sum = sum + k;
        }
        System.out.~~println~~ print("Student Completed" + this.classesCompleted + "classes");
        ~~with an average~~
        System.out.println("with an average of" + (sum / this.classesCompleted)f);
    }
}
```

12) (20 Marks)

Write a method that calculates postfix expressions. An example of a postfix expression is as follows:

Postfix
3 8 6 - * 2 / = 3

Your method should have the following signature:

```
public double calculatePostfix (Object[] equation)
```

The postfix equation will be stored in the array **equation**. Numbers are stored using the Double wrapper class and the operators (+,-,*,/) are stored using the Character wrapper class. The first cell of the array **equation** will contain the first number or operator. The end of the equation is marked by the first cell of the array that holds a null pointer or the end of the array if the equation takes up the entire array.

You also have access to the following methods from a stack class that stores stacks of double numbers:

Class DoubleStack

DoubleStack(): constructor, creates a new empty stack
double pop(): pops the top element from the stack and returns it. Throws an exception if the stack is empty.
push(double element): Pushes element onto the top of the stack
boolean isEmpty(): Returns true if the stack is empty
int size(): Returns the number of items that are currently stored on the stack

You can adapt this algorithm for the stack found on page 312 of the text.

1. Initialize a stack of double numbers
2. do
   if (the next input is a number)
   Read the next input and push it onto the stack
   else
   {
   Read the next character which is an operation symbol (e.g. +,-,*/).
   Pop two numbers from the stack.
   Combine the two numbers with the operation (using the second number popped as the left operand)
   Push the result onto the stack
   }
   while (there is more of the expression to read);
3. At this point, the stack contains one number, which is the value of the expression.

```
public double calculatePostfix (Object[] equation) {
int i = 0
DoubleStack holder = new DoubleStack();

do
  if (equation[i] {
    holder.push(equation[i].doubleValue());
    i++;
  }
  else {
  char c1 = equation[i].charValue();
  int j = holder.pop();
  int k = holder.pop();
  int sum = K c1 j ;
  holder.push(sum);
  i++;
  }
while (i < equation.length)

return (holder.pop());

}
```

13) (17 Marks)

The priority Queue class is like the regular Queue class but items placed in the queue also have priorities. Items are removed from the queue first by order of highest to lowest priority and then by First In First Out (FIFO). Your priority queue class should be implemented using an array of ordinary queues. Your priority queue should store objects.

You may use the class Queue and the following of it's methods.

object getFront(): removes and returns the front item from the queue. Throws an exception if the queue is empty.
insert(object obj): inserts an item at the end of the Queue
boolean isEmpty(): returns true if the queue is empty and false if the queue is not empty.
int size(): returns the size of the queue

Complete the following implementation of the PriorityQueue Class by filling in the implementation of the specified methods. Priority should increase the higher the number. That is, a priority of 6 would be higher than a priority of 4.

```
class PriorityQueue {

 // instance variables
 private Queue[] data;    // array of queues
 private int manyItems;   //number of items in the Queue

 // constructor for the PriorityQueueClass. Priorities
 // indicates how many priority classes
 // that the Priority Queue will maintain
public ~~void~~ PriorityQueue(int priorities) {
    data = new Queue [priorities];
    manyItems = 0;
}

 // insert a object into the priority queue
 public void insert (Object obj, int priority) {
    ~~data[priority] = data[priority]~~
    data[priority].insert(obj);
    manyItems++;
}
```

```
// returns the item at the front of the Priority Queue
// and removes it from the Priority Queue
// Throws an exception if the Priority Queue is empty
public Object getFront(){
    ~~while ( data[(data.length-1)] != null~~
    Object answer;
    if ( manyItems == 0 )
        throw new NoSuchElementException ("Priority Queue underflow.");
    int i = 1; boolean done = false;
    ~~while (!data[(data.length-i)].isEmpty~~
    while (!done) {
      if ( data[(data.length - i)].isEmpty == false ) {
        answer = data[data.length-i].getFront();
      } done = true;
      else
        i++;
    }
}

// returns the number of objects in the priority Queue
public int size() {
    return manyItems;
}

// returns true if there are no objects in the Queue
public boolean isEmpty() {
    return ( manyItems == 0 )
}
}
```